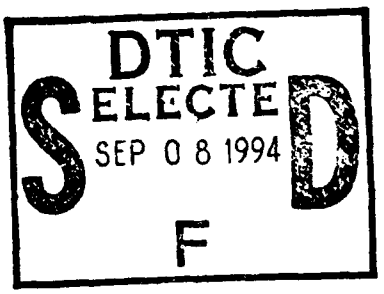
1

ARMY RESEARCH LABORATORY

Parallel Isosurface Generator for Huge Datasets

Kathy A. Burke

ARL-CR-171 August 1994



prepared by

Computer Sciences Corporation
3160 Fairview Park Drive
Falls Church, VA 22042

under contract

DAAL03-89-C-0038

NOTICES

Destroy this report when it is no longer needed. DO NOT return it to the originator.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute indorsement of any commercial product.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1994	Final, Jan-Dec 93

4. TITLE AND SUBTITLE

A Parallel Isosurface Generator for Huge Datasets

5. FUNDING NUMBERS

C: DAAL03-89-C-0038

6. AUTHOR(S)

Kathy A. Burke

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Army High Performance Computing Research Center
University of Minnesota
1100 Washington Avenue, South
Minneapolis, MN 55415

Computer Sciences Corporation
3160 Fairview Park Drive
Falls Church, VA 22042

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
ATTN: AMSRL-OP-AP-L
Aberdeen Proving Ground, MD 21005-5066

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

ARL-CR-171

11. SUPPLEMENTARY NOTES

The contract is between the University of Minnesota and the U.S. Government, subcontracted to Computer Sciences Corporation. Author is employed by Computer Sciences Corporation. Contracting Officer's Representative for this report is Dr. Walter B. Sturek, U.S. Army Research Laboratory, ATTN: AMSRL-CI-CA, Aberdeen Proving Ground, MD 21005-5067.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

A parallel C implementation of the marching cubes method is discussed. The input data for the code consists of a variation of the unformatted PLOT3D grid and solution files. The PLOT3D files can define an irregularly or regularly spaced mesh and the associated scalars given by v(x,y,z), F(x,y,z), and S_n = F(x,y,z), respectively. The output is a collection of three-sided polygons defining a constant value of F(x,y,z), and a second scalar which will be mapped onto this surface. The output is given in the *Bop* (Bag-O-Polygons) file format.

14. SUBJECT TERMS

marching cubes algorithm, parallel, *sproc*, *fork*, *pthread*, isosurface, parallel software, parallel C software, graphics, *Bop_View*, *Bop* format, SciAn, X-windows, GL, polygons

15. NUMBER OF PAGES

22

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

ACKNOWLEDGMENTS

This work was performed under the Army High Performance Computing Research Center (AHPCRC) Contract DAAL03-89-C-0038 with the University of Minnesota. The author would like to thank Mr. Jerry Clarke (AHPCRC/ Computer Sciences Corporation [CSC]) for his patient efforts and assistance, and Mrs. Deborah Thompson for her expert knowledge of Picsure in making Figures 4–6.

INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS

INTENTIONALLY LEFT BLANK.

LIST OF FIGURES

INTENTIONALLY LEFT BLANK.

1. INTRODUCTION

The birth of supercomputers and massively parallel processing machines affords researchers and scientists the computer power to simulate real world-class problems defined by huge grids. As a result of this, huge amounts of output data are generated; thereby creating a need for post-processing software capable of handling and evaluating such huge data sets.

This report will describe a parallel method of computing isosurfaces of a trivariate function, F(x,y,z), described by a variation of the unformatted PLOT3D file format. This output consists of three-sided polygons defining a surface where F(x,y,z) is some constant value. The format for the output file will be a generic polygonal format called Bag-O-Polygons (*Bop*). The purpose of a parallel version of the *marching cubes* method is to allow processing of huge data sets which cannot be read totally into a machine's memory. This *marching cubes* implementation is capable of computing isosurfaces of any size grid, whether it be regularly or irregularly spaced. This implementation relies on slicing the grid in the *k* direction, and therefore, would require the regular or irregular grid to be oriented in such a manner to accommodate this method. The C routines referenced in this report are capable of executing on the Silicon Graphics, Inc. workstations, Sun workstations, and the Kendall Square Research 1 machine. The multiple platform parallel execution is made possible through the use of sproc, fork, and pthreads. Timing results from the three platforms will be given in section 4 of this report.

2. MARCHING CUBES METHOD

Marching cubes is a high-resolution, three-dimensional (3D) surface construction algorithm written by Lorenson and Cline (1987). The *marching cubes* method is the means by which one defines a 3D surface of a constant value, S = F(x,y,z), from some volume of data represented by a structured grid (Figure 1). The term *marching* comes from the notion that one marches through the volume of data a *cube* at a time. A *cube* being a six-sided polygon defined by eight nodes and two adjacent planes in each direction of *i*, *j*, and *k*.

The algorithm marches through each *cube* determining whether the intended surface intersects any edges of the *cube*. The intersections are determined by placing a zero or a one at each of the eight nodes. A one is assigned to a *cube*'s node (vertex) if the data, S, at that node is greater than or equal to the user-specified value. Otherwise, a zero is placed at the node if S falls below the user-specified value. Using

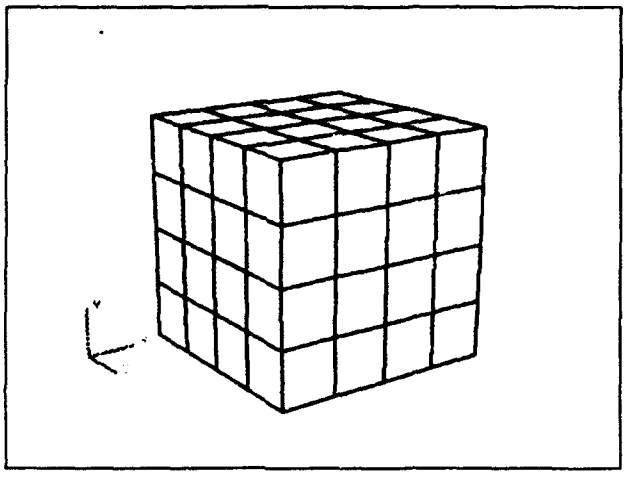


Figure 1. Structure grid.

this combination of zeros and ones, an 8-bit number is constructed. The 8-bit number is an index into a table containing 2^8 possible combinations of topologies for a surface defined within a given cube (Figure 2). Although there are 256 possible cases defining surface intersections with the *cube* edges; through symmetry operations and complementary cases, the problem can be reduced to 15 major cases. Figure 3 shows the 15 major cases which consist of 0–4 triangles per cube.

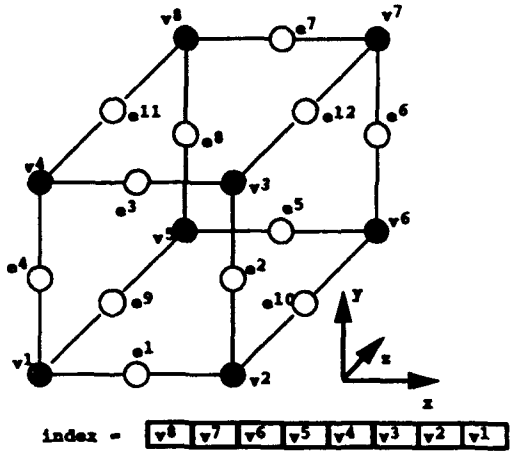


Figure 2. Cube indexing.

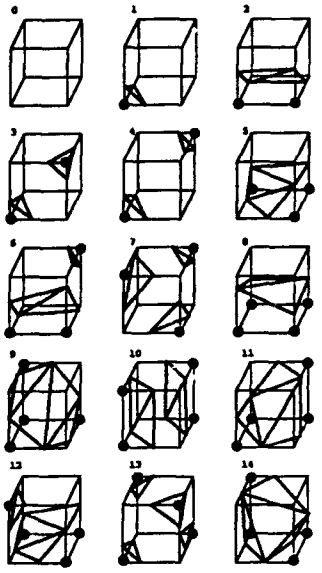


Figure 3. 15 case configuration.

After retrieving the triangulation for a specific *cube* from the table, linear interpolation is used to determine where the vertices of the triangle(s) will intersect with the edges of the *cube*. Thus, forming a list of three-sided polygons defining the connectivity of the isosurface.

3. PARALLEL IMPLEMENTATION

The parallel implementation of the *marching cubes* algorithm, written in C, runs on Silicon Graphics, Inc. (SGI) Workstations under IRIX,* on Sun Workstations under SunOS,** and on the KSR1 (Kendall Square Research 1) under KSR OS,*** which are all UNIX****-compatible.

3.1 Input Data Requirement. The parallel *marching cubes* code requires a variation of the C unformatted PLOT3D grid and solution files. More specifically, the parallel isosurface generator requests the input data in two separate PLOT3D multiple grid, 3D whole, xyz, and Q (without Jacobian) file formats.

---

* I and IRIX are trademarks of Silicon Graphics, Inc.
** n and SunOS are trademarks of Sun Microsystems.
*** SR1 and KSR OS are trademarks of Kendall Square Research Corporation.
**** UNIX is a trademark of Bell Laboratories.

Although the multiple grid format is used, the code expects the number of grids to equal one. Also, the Q file, otherwise referred to as the "solution file," is not limited to five specific scalar and vector components, as specified in the PLOT3D User's Manual (1989). The user will need at least one scalar defined in the solution file to complete a successful execution. In the case of one scalar, the user can define an isosurface of a constant value of that scalar, S, and map the identical scalar onto the surface for color mapping purposes. In instances when there are more than one scalar in the solution file the user has the flexibility to pick and choose any scalars for defining the isosurfaces and the isosurface mapping. See Appendix A for the description of the C syntax for writing this variation of the unformatted PLOT3D grid and solution files.

Additional input is required from the command line. The command line arguments will specify the PLOT3D grid, Plot3D solution, and *Bop* filenames for argv[1] through argv[3], respectively. The user should also supply the numeric index into the Q_file indicating which scalars, S_{iso} and S_{map}, will define the isosurface and isosurface mapping, argv[4] and argv[5]; a floating-point isosurface value, argv[6]; an integer value specifying the number of planes to step in the k direction, argv[7]; and finally, an integer value indicating the number of processes to spread the problem across, argv[8]. See Appendix B for an example shell script for executing this parallel isosurface generator using the above command line arguments.

3.2 Distribution of Workload. The parallel implementation consists of multiple routines; some routines are controlled by the parent process, while others are implemented by the children. The parent process controls the initial opening of the files to retrieve header information from both the grid and solution files. Once the number of grids (one) and dimensions in *i*, *j*, and *k* are read in, the parent calls the routine to partition the processing of the isosurface using the slice strategy. This partitioning routine creates a C structure containing the information each child requires in order to determine which part of the grid it will read and process. Having formed the partitioning structure, the parent produces children totaling the number specified by the command line argument, argv[8]. The parent creates new processes using *fork* or *sproc* on the SGI workstations, *fork* on the Sun workstations, and *pthreads* on the KSR1 machine. These new processes (children) are responsible for calculating polygons defining the isosurface, and writing the corresponding data to the *Bop* file. Due to the nature of the *marching cubes* problem, each child process may or may not be required to perform output operations. Therefore, children with small or no output will complete execution before those children with larger output requirements.

3.2.1 *Fork*. *Fork* is a standard UNIX call for creating a new process. This new process, known as the "child process", is an exact copy of the calling process (parent process). During a normal fork, the writable portions of the process's address space are marked copy-on-write. Hence, if any process writes onto a given page, a copy of that page is created and given to that process. Writes by one process are not visible to the other existing processes including the parent process (calling process). Although the child process is an exact copy of the parent process, the child has a unique process id (pid), a different parent process ID, and has its own copy of the parent's file descriptors.

3.2.2 *Sproc*. *Sproc* is a SGI, IRIX specific routine which creates a new share group process. While *sproc* is a variant of the standard UNIX *fork* call, inherent differences between the two exist. When a parent or child calls *sproc*, a new process is created. Instead of the new process being an exact copy of the parent, as in *fork*, the child shares the virtual address space (shared memory, mapped files, and data space) of the parent process. This is, of course, assuming that one has selected the sharing option. In the case of *sproc*, the parent and child each have their own stack pointer and program counter, but all the data and text space is visible by both processes. After a successful *sproc*, the parent and child process will have unique pids, but are in the same shared process group.

The first time *sproc* is called, a share group or shared process group is formed. All subsequent calls to *sproc*, whether it be by the parent or child, will add another process to the share group. As mentioned above, all members of the share group, share virtual address space, as well as possible sharing of file tables, effective userids, current working directories, and other options which may be specified by the *inh* flag of *sproc*.

3.2.3 *Pthreads*. The KSR *pthread* is a high-level interface to the IEEE POSIX thread library (IEEE 1990). KSR C *pthread* (POSIX threads) functions are called to create and synchronize processes. *Pthread* objects are made up of *pthreads* themselves, mutexes, condition variables, and barriers. Variables accessible to one *pthread* are available to all other *pthreads* in that process. Multiple threaded processes operate in a single, shared address space. Due to the sharing of address space, creating *pthreads* incurs considerably less overhead than creating a new process. In general, multi-threaded processes share most of their data, but often use private variables for thread operations.

Pthreads use the qualifiers, *_private* and *_shared*, to specify whether a declared variable will be pthread-private or shared. In declaring a variable pthread-private, each *pthread* has a private copy of that

variable. All variables not declared private are shared; therefore, the qualifier *_shared* is never required. Default private variables, such as automatic and register variables cannot be declared shared.

3.3 Data Output Format. Although there are many output formats which can be defined for a large variation of commercial graphics packages, this parallel isosurface generator outputs the *Bop* format. The *Bop* format can be easily written and networked with the use of the *Bop* libraries. These libraries have been linked with the parallel isosurface code to allow the polygonal data to be written to disk files or networked across heterogeneous architectures using *Bop_View* (Clarke 1993).

Since this parallel isosurface generator utilizes the slice strategy for implementing parallelism, some, if not all of the child processes could very well arrive at the function responsible for writing polygons in the *Bop* format, simultaneously. To prevent such collisions, semaphores have been employed to allow only one child to write or network polygons in the *Bop* format at a time.

Upon completion of the isosurface generation, the *Bop* format in conjunction with *Bop_View* saves images from the viewport in a number of file formats. These files can later be used to create 3D video animations or color hardcopies. Refer to the *Bop* and *Bop_View* documentation (Clarke 1993) for a complete description and examples for using this file format and application.

3.3.1 *Bop* Format. The *Bop* format is a binary, polygonal format. This format contains the information necessary for *Bop_View* to visualize the list of polygons defining the isosurface. Each polygon is defined by a C structure, *bpoly*, whose members include: an integer specifying the number of vertices and the floating-point values for x, y, z, and the mapping scalar for each vertex of the polygon. The *libbop.a* library contains the functions required for opening, writing to, and closing *Bop* files.

It is also important to note that the Florida State University and the Supercomputer Research Institute's public domain scientific visualization and animation package, SciAn, has a *Bop* file reader.

3.3.2 *Bop_View.* This is an X-window, Motif application program for visualizing polygonal data. *Bop_View's* X-window viewport is capable of rendering polygons to an X-window, with the option of rendering these polygons in the SGI Graphics Language (GL). Once the polygons have been rendered in the viewport, the user can use combinations of x, y, z translations, rotations, and scaling to orient the isosurface to a desirable location and size. The user has the choice of writing the polygons in SGI's rgb

(red green blue), BRL_CAD's pix, or PostScript's file format. The *libbop_mrs.a* library contains the functions required for opening and sending polygonal information via a TCP/IP connection to a *Bop_View* process locally or remotely.

4. RESULTS

The parallel implementation of the *marching cubes* algorithm has been used to generate isosurfaces of computational data from the CTH and HULL codes, as well as medical resonance data. All three output formats were converted to a variation of the PLOT3D grid and solution file formats, as mentioned earlier.

Prior to the development of this parallel isosurface generator, a 3D volume of computational data of ~2.2 M floating-point grid values and ~11 M floating-point scalar values required a wallclock time of ~20 min per time-step to read the entire grid and associated scalars into memory, calculate the desired isosurface of ~60 thousand polygons, write the polygons to an X-window, and dump the viewport of the X-window into an image file. Using the same dataset while processing the isosurface with the parallel isosurface generator, writing polygons via MRS to *Bop_View*, and generating an image file takes ~90 s.

Figures 4–6 represent timing results of isosurface generation using the parallel implementation on the SUN4m sparc, SGI Challenge, and KSR1 machines. Timings are based on an eight processor maximum for the SGI and the KSR1, and two processors for the SUN4 sparc. All timings include total execution time, which includes the time for input and output. Therefore, timing differences between the KSR1 and the SGI and SUN4m platforms are considerably different due to input/output contingencies on the KSR1. The timing results are given to simply show different test cases and are not being used as an evaluation of any of the specific architectures.

5. CONCLUSIONS

Huge datasets, whether they be on the order of millions or billions of points, can be post processed with this parallel isosurface generator, given that at least two slices (marching in the k-direction) can fit into a machine's physical memory. This piecewise calculation has been tested on SUN4, SGI, and KSR1 machines, yielding results not possible using other commercial and public domain packages which assume that one's volume of data will fit entirely into a machine's memory.

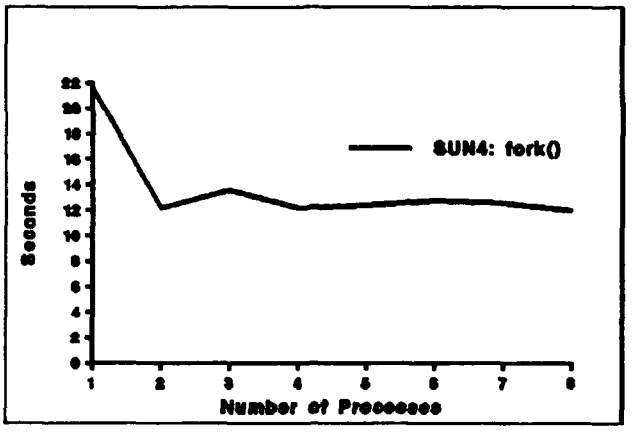


Figure 4. Time vs. number of processors plot for SUN4.

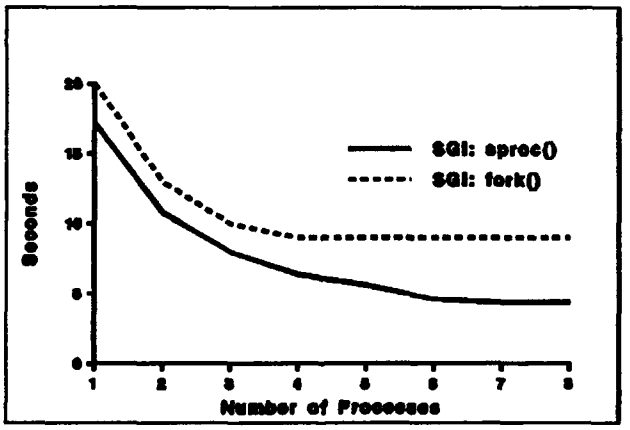


Figure 5. Time vs. number of processors plot for SGI

Linking this parallel isosurface generator with a given computation code can yield a machine-independent means for creating interactive graphics. The combination of the piecewise isosurface computation and the X-window display via TCP/IP using *Bop_View*, makes for a flexible environment for producing interactive visualization. This capability is a useful tool for analyzing and debugging 3D volumetric data.

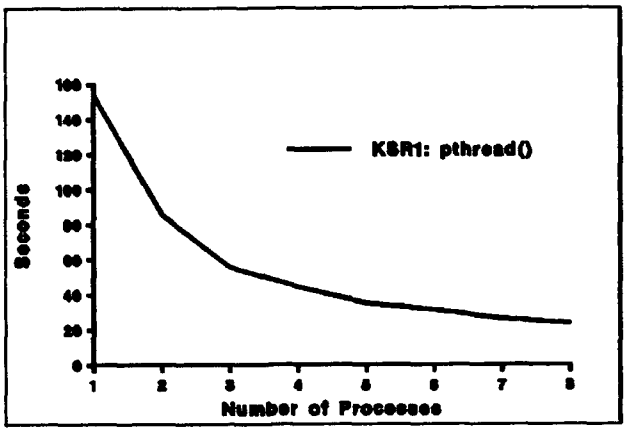


Figure 6. Time vs. number of processors plot for KSR1.

INTENTIONALLY LEFT BLANK.

6. REFERENCES

Clarke, J. "Distributed Heterogeneous Visualization: *Bop* and *Bop-View*." U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, to be published.

Lorensen, W. E., and H. E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm." SIGGRAPH 87 Conference Proceedings, Computer Graphics, vol. 21, no. 4, pp. 163–169, July 1987.

NASA Ames Research Center. PLOT3D User's Manual. Moffett Field, CA, 1989.

Institute of Electrical and Electronics Engineers. Threads Extension for Portable Operating Systems. (P1003.4a), p. 10, 1989.

INTENTIONALLY LEFT BLANK.

APPENDIX A:

UNFORMATTED PLOT3D VARIATION OF THE GRID AND SOLUTION FILES

INTENTIONALLY LEFT BLANK.

A. Unformatted PLOT3D Variation Of The Grid And Solution Files

```
/*  This Function Reads A Generic Data File And Writes Two Files Which Are
    A Variation Of The PLOT3D Multiple Grid (3D Whole, XYZ) And Q (Without
    Jacobian) File Formats.  */

#include <stdio.h>
void
var_plot3d(char **argv)
{
int       ngrids = 0;
int       idim = 0, jdim = 0, kdim = 0;
int       num_verts = 0;
int       i = 0;
int       proc_id = 1;
int       num_forks = 2;
int       fork_num = 0;
int       status = 0;
int       child[2] = {0, 0};

float     x = 0.0, y = 0.0, z = 0.0;
float     w = 0.0, rha = 0.0, rho = 0.0, pres = 0.0;
float     *vx = NULL, *vy = NULL, *vz = NULL;
float     *vw = NULL, *vrha = NULL, *vrho = NULL, *vpres = NULL;
float     scalar[4] = {0, 0, 0, 0};

FILE      *input = NULL, *grid = NULL, *sol = NULL;


/* Open Input And Output Files Given From The Command Line. */

input = fopen(argv[1],"r");
grid  = fopen(argv[2],"w");
sol   = fopen(argv[3],"w");


/* Grab Dimension In The I, J, and K Direction From the Command Line. */

idim  = atoi(argv[4]);
jdim  = atoi(argv[5]);
kdim  = atoi(argv[6]);

ngrids    = 1;
num_verts = idim * jdim * kdim;


/* Allocate Memory For Grid And Solution Variables. */

if((vx = (float * )calloc(num_verts,sizeof(float))) == NULL){
    fprintf(stderr,"Unable To Calloc vx.\n");
    perror("Calloc :");
    exit(-1);
    }

if((vy = (float *)calloc(num_verts,sizeof(float))) == NULL){
  fprintf(stderr,"Unable To Calloc vy.\n");
  perror("Calloc :");
  exit(-1);
  }

if((vz = (float *)calloc(num_verts,sizeof(float))) == NULL){
  fprintf(stderr,"Unable To Calloc vz.\n");
```

```
  perror("Calloc :");
  exit(-1);
  }

if((vw = (float * )calloc(num_verts,sizeof(float))) == NULL){
    fprintf(stderr,"Unable To Calloc vw.\n");
    perror("Calloc :");
    exit(-1);
    }

if((vrha = (float *)calloc(num_verts,sizeof(float))) == NULL){
  fprintf(stderr,"Unable To Calloc vrha.\n");
  perror("Calloc :");
  exit(-1);
  }

if((vrho = (float *)calloc(num_verts,sizeof(float))) == NULL){
  fprintf(stderr,"Unable To Calloc vrho.\n");
  perror("Calloc :");
  exit(-1);
  }

if((vpres = (float *)calloc(num_verts,sizeof(float))) == NULL){
  fprintf(stderr,"Unable To Calloc vpres.\n");
  perror("Calloc :");
  exit(-1);
  }


/* Read Data From A Generic Data File. */

for(i=0;i<num_verts;i++){
  fscanf(input,"%f %f %f", &x, &y, &z);
  fscanf(input,"%f %f", &w, &rha);
  fscanf(input,"%f %f", &rho, &pres);

  vx[i] = x;
  vy[i] = y;
  vz[i] = z;

  vw[i]    = w;
  vrha[i]  = rha;
  vrho[i]  = rho;
  vpres[i] = pres;

  }


/* Fork Two Processes, One Will Be Used To Create The Grid, And The Other For
   The Solution. */

for(i=0; i<num_forks; i++){
  fork_num = i;

  if(proc_id!=0){
    proc_id = fork();
    }
  else{
    break;
    }

  if(proc_id == -1){
    perror("fork");
    fprintf(stderr,"ERROR - CAN NOT FORK PROCESS.\n");
    exit(-1);
```

```
    } /* end if proc_is == -1 */
  else if(proc_id == 0){
    if(fork_num == 0){
      fwrite(&ngrids,sizeof(int),1,grid);
      fwrite(&idim,sizeof(int),1,grid);
      fwrite(&jdim,sizeof(int),1,grid);
      fwrite(&kdim,sizeof(int),1,grid);
      fwrite(vx,sizeof(float),num_verts,grid);
      fwrite(vy,sizeof(float),num_verts,grid);
      fwrite(vz,sizeof(float),num_verts,grid);
      child[fork_num] = getpid();
      printf("child[%d] = %d, process completed!\n",fork_num,child[fork_num]);
      } /* end if fork_num == 0 */
    if(fork_num == 1){
      fwrite(&ngrids,sizeof(int),1,sol);
      fwrite(&idim,sizeof(int),1;sol);
      fwrite(&jdim,sizeof(int),1,sol);
      fwrite(&kdim,sizeof(int),1,sol);
      fwrite(scalar,sizeof(float),4,sol);
      fwrite(vw,sizeof(float),num_verts,sol);
      fwrite(vrha,sizeof(float),num_verts,sol);
      fwrite(vrho,sizeof(float),num_verts,sol);
      fwrite(vpres,sizeof(float),num_verts,sol);
      child[fork_num] = getpid();
      printf("child[%d] = %d, process completed!\n",fork_num,child[fork_num]);
      } /* end if fork_num == 1 */
    } /* end if proc_id == 0 */
  } /* end for i */

/*  Have Parent Process Wait Until The Children Have Completed. */

for(i=0;i<num_forks;i++){
  wait(&status);
  }

if(proc_id !=0){
  exit(num_forks);
  }
}
```

INTENTIONALLY LEFT BLANK.

APPENDIX B:

PARRALLEL ISOSURFACE GENERATOR SHELL SCRIPT

INTENTIONALLY LEFT BLANK.

B. Parrallel Isosurface Generator Shell Script

```
#! /bin/csh -f

#  This shell script executes PBIG with the following parameters:
#         Grid       File Name: /usr/tmp/Kburke/hdf02.grid
#         Solution   File Name: /usr/tmp/Kburke/hdf02.sol
#         BOP        File Name: /usr/tmp/Kburke/bop1
#         Scl_num_iso_data  = 2
#         Scl_num_iso_map   = 1
#         Isovalue          = 5.0
#         Kstep             = 3
#         Num_procs         = $1
#
#  Note:  User should set Kstep equal to 2 or 3
#
if($1 == '')then
echo "Usage: $0 Num_procs"
exit (1)
else
set DDIR1=/usr/tmp/Kburke
set GRID=hdf02
set SOL=hdf02
endif

rm -r $DDIR1/bop1

echo "Executing /usr/tmp/Kburke/PBIG"
/usr/tmp/Kburke/PBIG $DDIR1/$GRID.grid $DDIR1/$SOL.sol $DDIR1/bop1 2 1 5.0 3 $1
```

INTENTIONALLY LEFT BLANK.

No. of Copies	Organization
2	Administrator Defe e Technical Info Center ATT DTIC-DDA Came on Station Alexandria, VA 22304-6145
1	Commander U.S. Army Materiel Command ATTN: AMCAM 5001 Eisenhower Ave. Alexandria, VA 22333-0001
1	Director U.S. Army Research Laboratory ATTN: AMSRL-OP-CI-AD, Tech Publishing 2800 Powder Mill Rd. Adelphi, MD 20783-1145
1	Director U.S. Army Research Laboratory ATTN: AMSRL-OP-CI-AD, Records Management 2800 Powder Mill Rd. Adelphi, MD 20783-1145
2	Commander U.S. Army Armament Research, Development, and Engineering Center ATTN: SMCAR-TDC Picatinny Arsenal, NJ 07806-5000
1	Director Benet Weapons Laboratory U.S. Army Armament Research, Development, and Engineering Center ATTN: SMCAR-CCB-TL Watervliet, NY 12189-4050
1	Director U.S. Army Advanced Systems Research and Analysis Office (ATCOM) ATTN: AMSAT-R-NR, M/S 219-1 Ames Research Center Moffett Field, CA 94035-1000

No. of Copies	Organization
1	Commander U.S. Army Missile Command ATTN: AMSMI-RD-CS-R (DOC) Redstone Arsenal, AL 35898-5010
1	Commander U.S. Army Tank-Automotive Command ATTN: AMSTA-JSK (Armor Eng. Br.) Warren, MI 48397-5000
1	Director U.S. Army TRADOC Analysis Command ATTN: ATRC-WSR White Sands Missile Range, NM 88002-5502
(Class. only) 1	Commandant U.S. Army Infantry School ATTN: ATSH-CD (Security Mgr.) Fort Benning, GA 31905-5660
(Unclass. only) 1	Commandant U.S. Army Infantry School ATTN: ATSH-WCB-O Fort Benning, GA 31905-5000
	Aberdeen Proving Ground
2	Dir, USAMSAA ATTN: AMXSY-D AMXSY-MP, H. Cohen
1	Cdr, USATECOM ATTN: AMSTE-TC
1	Dir, USAERDEC ATTN: SCBRD-RT
1	Cdr, USACBDCOM ATTN: AMSCB-CII
1	Dir, USARL ATTN: AMSRL-SL-I
5	Dir, USARL ATTN: AMSRL-OP-AP-L

No. of
Copies Organization

1 Computer Sciences Corporation
ATTN: Dr. David Brown
3160 Fairview Park Dr.
Mail Code 265
Falls Church, VA 22042

Aberdeen Proving Ground

11 Dir, USARL
ATTN: AMSRL-CI, William Mermagen
AMSRL-CI-A, Harold Breaux
AMSRL-CI-AC,
John Grosh
Phillip Dykstra
Jerry Clarke
Deborah Thompson
Jennifer Hare
Eric Mark
Richard Angelini
Kathy Burke
AMSRL-CI-C, Walter Sturek

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number ARL-CR-171 Date of Report August 1994

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)

DEPARTMENT OF THE ARMY

OFFICIAL BUSINESS

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 0001, APG, MD

Postage will be paid by addressee

Director
U.S. Army Research Laboratory
ATTN: AMSRL-OP-AP-L
Aberdeen Proving Ground, MD 21005-5066